

APPENDIX I

```
% Creating a network topology object
5   network_topo = topo('init');           % graphically place nodes on screen
    addlink(network_topo);                 % graphically connect up nodes
    labelnames(network_topo);              % graphically label nodes
10  save network_topo;                   % save network_topo for future use

% Top level procedure to compute paths that optimize use of network capacity
% inputs:
15  %     D = traffic demand matrix
    %     (retrieved from predictions stored in TMS Statistics Repository)
    %     network_topo = topology object defining the network topology
    %     P = network policy information
    %     (matrix of reserved capacity, which indicates links whose use
    %     is administratively prohibited or which should not be
    %     completely allocated)
    % outputs:
    %     allocated_paths() = list of paths to set up, to TMS signalling system

20
25  C = capacity(network_topo);          % retrieve network topology information
    C = C - P;

    saved_C = [];
    saved_SLA = [];
    assigned_paths = [];
    round = 0;

30
35  [SLA, S] = create_ordered_sla(D);
    F = SLA(1);

    for F = SLA';
        round = round +1;
        saved_C{round} = C;
        saved_SLA{round} = F;
40
45  F % display the flow

        W = calc_weights('calcweight2',F,C);
        [dist, P] = floyd(W);
```

```

path = findpath(P,F.i,F.j);

5      assigned_paths{round}.path = path;
      assigned_paths{round}.flow = F;

if isempty(path)
    fprintf(1,'no path for flow:\n'); F
else
10     C = compute_residual_capacity('c - F.bw',path,F,C);
end

end

15 function [W] = calc_weights(func,F,C)
% function [W] = calc_weights(func,F,C)
%
% Compute the weights by calling func on each elt of C
% func must be of the form double func(Flow F, Capacity_elt c, node i, node j)

20 func = fcnchk(func);

for i = 1:size(C,1)
    for j = 1:size(C,2)
25        W(i,j) = feval(func,F,C(i,j),i,j);
    end
end

30 function [w] = calcweight2(F,c,i,j)
% function [w] = calcweight2(F,c,i,j)
% basic weight calc

if (0 == c)
    w = inf;
35    return;
end

40 % rule out paths that can't hack it
if (F.bw > c)
    w = inf;
    return;
end

45

```

```

w = 1 / (c - F.bw); % fill links with most capacity first

function [C] = compute_residual_capacity(func, path, F, C)
% function [C] = compute_residual_capacity(func, path, F, C)
5
% Update capacity characteristics in C to reflect flow F being
% allocated along path using function func
% func should be of the form
%   C_element func(C_element c, Flow F)

10

if (length(path) <= 1)
    return;
end

15
func = fcchk(func,'c','F');

index = 1;
src = path(index);
index = index + 1;

20

for index = index:length(path)
    dst = path(index);

    C(src,dst) = feval(func,C(src,dst),F);

    src = dst;
end

25

function [SLA, S] = create_ordered_sla(D)
% function [SLA] = create_ordered_sla(D)
% takes the demand matrix and returns a list of SLAs,
%   SLA of the form [ struct ; struct ; ... ] where struct is [BW, i, j]
%   S of the form [ [BW, i, j] ; [BW, i, j] ; ... ]

30

S = [];

35

for i = 1:size(D,1)
    for j = 1:size(D,2)
        if (D(i,j) ~= 0)
            S = [[D(i,j) i j] ; S];
        end
    end
end

40

45

```

```

[Y, I] = sortrows(S,1);

S = Y(size(Y,1):-1:1,:); % reverse order
5
SLA = struct('bw',num2cell(S(:,1)), 'i',num2cell(S(:,2)), 'j',num2cell(S(:,3)));
return;

10 function [path] = findpath(P,i,j)
% function [path] = findpath(P)
%
%
15 path = [];

if (i == j)
    path = [i];
    return;
end

20 if (0 == P(i,j))
    path = [];
else
    path = [findpath(P,i,P(i,j)) j];
end

25 function [D, P] = floyd(W)
% function [D, P] = floyd(W)
% given weights Wij, compute min dist Dij between node i to j
% on shortest path from i to j, j has immeadiate predecessor Pij

30 n = size(W,1);
if (n ~= size(W,2))
    error('Input W is not square??!!');
end

35 D = W;

40 P = repmat([1:n]',[1 n]);
P = P .* ~isinf(W);
P = P .* ~eye(n);

45 for k = 1:n
        for i = 1:n

```

```
for j = 1:n
    alt_path = D(i,k) + D(k,j);
    if (D(i,j) > alt_path)
        D(i,j) = alt_path;
        P(i,j) = P(k,j);
    end
end
5
k;
10
D;
P;
end
```

APPENDIX II

```

function addlink(TOPO)
% addlink(TOPO)
%
5    % interactively add links to the TOPO

update(TOPO);
c_src = 1;
c_dst = 2;
10   c_bw = 3;

figure(TOPO.cur_fig)

while (1)
15
    fprintf(1,'n\nHit Button 3 to end...n\n');

    % find coords and index i of src
    [x1i y1i button] = ginput(1);
    if (button == 3) break; end

    d = sqrt((TOPO.locs(:,1) - x1i).^2 + (TOPO.locs(:,2) - y1i).^2);
    [d,i] = min(d);
    x1 = TOPO.locs(i,1); y1 = TOPO.locs(i,2);

    % find coords and index j of dst
    [x2i y2i] = ginput(1);
    d = sqrt((TOPO.locs(:,1) - x2i).^2 + (TOPO.locs(:,2) - y2i).^2);
    [d,j] = min(d);
    x2 = TOPO.locs(j,1); y2 = TOPO.locs(j,2);

    hold on;
    lh = line([x1 x2],[y1 y2],'color','red');

35
    cap = input('Enter capacity (in Mbps) > ');

    fprintf(1,'About to create symmetric %d Mbps link from node %d to node %dn',cap,i,j);

    doit = input('Enter Y to confirm, N to reject, and B to change bandwidth (Y)> ','s');

40
    if (isempty(doit)) doit = 'Y'; end

    if (doit == 'n' | doit == 'N')
        delete(lh);
        return;
45

```

```

end

if (doit == 'b' | doit == 'B')
    buf = sprintf('Enter capacity from %d to %d (in Mbps) >',i,j);
    cap_i_to_j = input(buf);

    buf = sprintf('Enter capacity from %d to %d (in Mbps) >',j,i);
    cap_j_to_i = input(buf);
else
    cap_i_to_j = cap;
    cap_j_to_i = cap;
end

%build the link records
clear linkab linkba;

linkab.src = i;
linkab.dst = j;
linkab.bw = cap_i_to_j;
linkab.handle = lh;

linkba.src = j;
linkba.dst = i;
linkba.bw = cap_j_to_i;
linkba.handle = lh;

% now draw the actual link on the map
delete(lh);
lh = drawlink(TOPO, linkab);

% now store the link info
TOPO.links = [TOPO.links ; linkab ; linkba ];
TOPO.linkarray = [TOPO.linkarray ; [ i j cap_i_to_j ] ; [ j i cap_j_to_i ]];
35
end % of while loop

assignin('caller',inputname(1),TOPO);

40

function [C, portmap] = capacity(TOPO)
45
% [C, portmap] = capacity(TOPO)
```

```

% portmap maps indices of C to elts of nodes(TOPO)
% [node dir] where
%     node is index of elt in nodes(TOPO)
%     dir is 1 if data enters here, -1 if data leaves here
5
numnodes = length(TOPO.links) * 2;

C = zeros(numnodes,numnodes);

10    curnode = 0;
portmap = [];
for i = 1:length(TOPO.links)
    link = TOPO.links(i);
    curnode = curnode + 1;
    15    portmap(curnode,:) = [link.src -1];
    curnode = curnode + 1;
    portmap(curnode,:) = [link.dst 1];

    C(curnode-1,curnode) = link.bw;
end

20    c_node = 1;
c_dir = 2;

25    for i = 1:length(TOPO.nodes)
        ins = find(portmap(:,c_node) == i & portmap(:,c_dir) == 1);
        outs = find(portmap(:,c_node) == i & portmap(:,c_dir) == -1);

        for j = ins
            for k = outs
                C(j,k) = inf;
            end
        end
    end
35    function [a, b, c] = debug(t)

        update(t);

        fieldnames(t)

40    a = t.nodes
    b = t.locs
    c = t.links
    function display(TOPO)
        45    % DISPLAY a topo object

```

```

% a link is a unidirectional, so the value is probably twice what you want

5   fprintf(['TOPO object: %d nodes %d links]\n',...
           length(TOPO.nodes),length(TOPO.links));
function draw(TOPO)
% draw(topo)
%
% draw the topology figure in a new window

10  TOPO.cur_fig = figure;
    axis(TOPO.axis);
    axis equal;
    axis manual;
    box on;
    hold on;

15

for i = 1:length(TOPO.nodes)
    nm = plot(TOPO.nodes{i}.loc(1),TOPO.nodes{i}.loc(2),'ob');
    TOPO.nodes{i}.mark_handle = nm;
    if (isfield(TOPO.nodes{i},'nameloc'))
        TOPO.nodes{i}.nameloc(3) = text(TOPO.nodes{i}.nameloc(1),...
                                         TOPO.nodes{i}.nameloc(2),TOPO.nodes{i}.name);
    end
end

20

25  % yes, this draws the same link twice. fix it if it matters -dam 11/21

TOPO.linkarray = [];
for i = 1:length(TOPO.links)
    TOPO.links(i).handle = drawlink(TOPO,TOPO.links(i));
    TOPO.linkarray = [TOPO.linkarray ; ...
                     [ TOPO.links(i).src TOPO.links(i).dst TOPO.links(i).bw]];
end

30

35  assignin('caller',inputname(1),TOPO);
function ex(t)

t.nodes

40  function labelnames(TOPO)
% function labelnames(TOPO)
% make it easy to label the nodes

45  for i = 1:length(TOPO.nodes)
      fprintf('Place label for node %d "%s"\n',i,char(TOPO.nodes{i}.name));

```

```

origcolor = get(TOPO.nodes{i}.mark_handle,'color');
set(TOPO.nodes{i}.mark_handle,'color',[1 0 0]);

5      if (isfield(TOPO.nodes{i},'nameloc'))
          good_x = TOPO.nodes{i}.nameloc(1);
          good_y = TOPO.nodes{i}.nameloc(2);
      end
      th = [];
      while (1)
          10     fprintf('Button 1 to (re)place text, Button 3 to accept\n');
          [x,y,button] = ginput(1);
          if (3 == button) break; end
          if (~isempty(th)) delete(th); end
          th = text(x,y,TOPO.nodes{i}.name);
          15     good_x = x; good_y = y;
          end
          TOPO.nodes{i}.nameloc = [good_x, good_y, th];
          set(TOPO.nodes{i}.mark_handle,'color',origcolor);
      end

assignin('caller',inputname(1),TOPO);function names(TOPO)
% NAMES the list of names of the nodes in the topo

fprintf('Node\tName\n');
25      for i = 1:size(TOPO.names,1)
          fprintf('%d\t%s\n',i,TOPO.names{i});
      end
      function [node] = nodes(TOPO)
% function [node] = nodes(TOPO)
%   returns a cell array describing nodes in the TOPO

35      node = TOPO.nodes;
      function [TOPO] = topo(TOPO)
% [TOPO] = topo(TOPO)
% if input TOPO is 'init', create a new topology
%
%       newtopo = topo('init');
%
% else add new nodes to TOPO
%
40      % nodes is a array of structs, one per node
% link is a array of structs, one per link
%   a link is a unidirectional item, so there are probably twice
%   as many links as you'd expect.

```

```

if (nargin < 1)
    error('topo(TOPO) or topo("init") - not enough args');
end

5      if (ischar(TOPO) & TOPO == 'init')
            clear TOPO

            TOPO.nodes = [];
            TOPO.links = [];

10     TOPO.capacity = []; % now computed as needed
            TOPO.locs = []; % internal cache
            TOPO.linkarray = []; % internal cache

15     f = figure;
            axis([0 75 0 50]);
            TOPO.axis = axis;
            TOPO.cur_fig = f;
            axis equal
            axis manual
            box on
            hold

20     else
            figure(TOPO.cur_fig);
        end

25

30     nodecount = length(TOPO.nodes);

while (1)
    clear nodeinfo;
    fprintf(1, '\n\nHit Button 3 to stop\n\n');
    [x y but] = ginput(1);
    if (but == 3) break; end
    x = floor(x); y = floor(y);
    nm = plot(x,y,'ob');
    name = input('Enter name > ','s');

35

40     nodeinfo.loc = [ x y];
     nodeinfo.mark_handle = nm;
     nodeinfo.name = cellstr(name);
     nodecount = nodecount + 1;
     TOPO.nodes{nodecount} = nodeinfo;

45

```

```

if ('topo' ~= class(TOPO))
    TOPO = class(TOPO,'topo');
end
5
if (nargout == 0)
    assignin('caller',inputname(1),TOPO);
end
function lh = drawlink(TOPO, link)
% assumes TOPO.linkarray is already valid, and draws the position of
% link line based on the number of links already present in linkarray

c_src = 1;
c_dst = 2;
15    c_bw = 3;

i = link.src;
j = link.dst;

x1 = TOPO.nodes{i}.loc(1);
y1 = TOPO.nodes{i}.loc(2);
x2 = TOPO.nodes{j}.loc(1);
y2 = TOPO.nodes{j}.loc(2);

25
if (isempty(TOPO.linkarray))
    num_links = 0;
else
    num_links = sum(TOPO.linkarray(:,c_src) == i & TOPO.linkarray(:,c_dst) == j);
end
30
pattern = [ 0 1 -1 2 -2 3 -3] * .3;

if (abs(x1 - x2) > abs(y1 - y2))
    delta_x = 0;
    delta_y = pattern(num_links + 1);
35
else
    delta_x = pattern(num_links + 1);
    delta_y = 0;
end
40
lh = line([x1 x2] + delta_x, [y1 y2] + delta_y, 'color', 'black');

45
function update(TOPO)

```

```
clear TOPO.locs;
for i = 1:length(TOPO.nodes)
    TOPO.locs(i,:) = TOPO.nodes{i}.loc
end
5

clear TOPO.linkarray;
for i = 1:length(TOPO.links)
    TOPO.linkarray = [TOPO.linkarray ; ...
        [ TOPO.links(i).src TOPO.links(i).dst TOPO.links(i).bw]];
end
10

% these are here to be cut and pasted into other functions as needed
% there doesn't seem to be a good way to pass them around in another fashion
% (using assignin('caller')...) to force their definition sounds like asking
15
% for trouble 'cause you'll overwrite another definition of them...)
c_src = 1;
c_dst = 2;
c_bw = 3;

assignin('caller',inputname(1),TOPO);
20
```